Two-player Games – Part 2

ZUI 2018/2019

Branislav Bošanský

bosansky@fel.cvut.cz

Previously … on Two-Player Games

- minimax search
- alpha-beta pruning
- extensive-form games

… and now

- Negascout
- Monte Carlo Tree Search
- Game Theory

Alpha-Beta Pruning

- **function** alphabeta(node, depth, α, β, Player)
- **if** (depth = 0 or node is a terminal node) **return** evaluation value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- α := max(α, alphabeta(child, depth-1, α, β, switch(Player)))
- **if** (β≤α) break
- **return** α
- **else**
- **for each** child of node
- β := min(β, alphabeta(child, depth-1, α, β, switch(Player)))
- **if** (β≤α) break
- **return** β

Negamax

- **function** negamax(node, depth, α, β, Player)
- **if** (depth = 0 or node is a terminal node) **return** the heuristic value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- α := max(α, -negamax(child, depth-1, -β, -α, switch(Player)))
- **if** (β≤α) break
- **return** α
- **else**
- **for each** child of node
- β := min(β, alphabeta(child, depth-1, α, β, not(Player)))
- **if** (β≤α) break
- **return** β

Aspiration Search

- [α, β] interval – window
- alphabeta initialization [-∞, +∞]

Aspiration Search

- $[\alpha, \beta]$ interval – window
- alphabeta initialization $[-\infty, +\infty]$
- what if we use $[\alpha_0, \beta_0]$
  - x = alphabeta(node, depth, α_0, β_0,player)
  - $\alpha_0 \leq x \leq \beta_0$ - we found a solution
  - $x \leq \alpha_0$ - failing low (run again with $[-\infty, x]$)
  - $x \geq \beta_0$ - failing high (run again with $[x, +\infty]$)

NegaScout – Main Idea

- enhancement of alpha-beta algorithm
- assume some heuristic that determines move ordering
  - the algorithm assumes that the first action is the best one
  - after evaluating the first action, the algorithm checks whether the remaining actions are worse
  - the “test” is performed via null-window search

Scout –Test

- what we really need at that moment is a bound (not the precise value)

Scout –Test

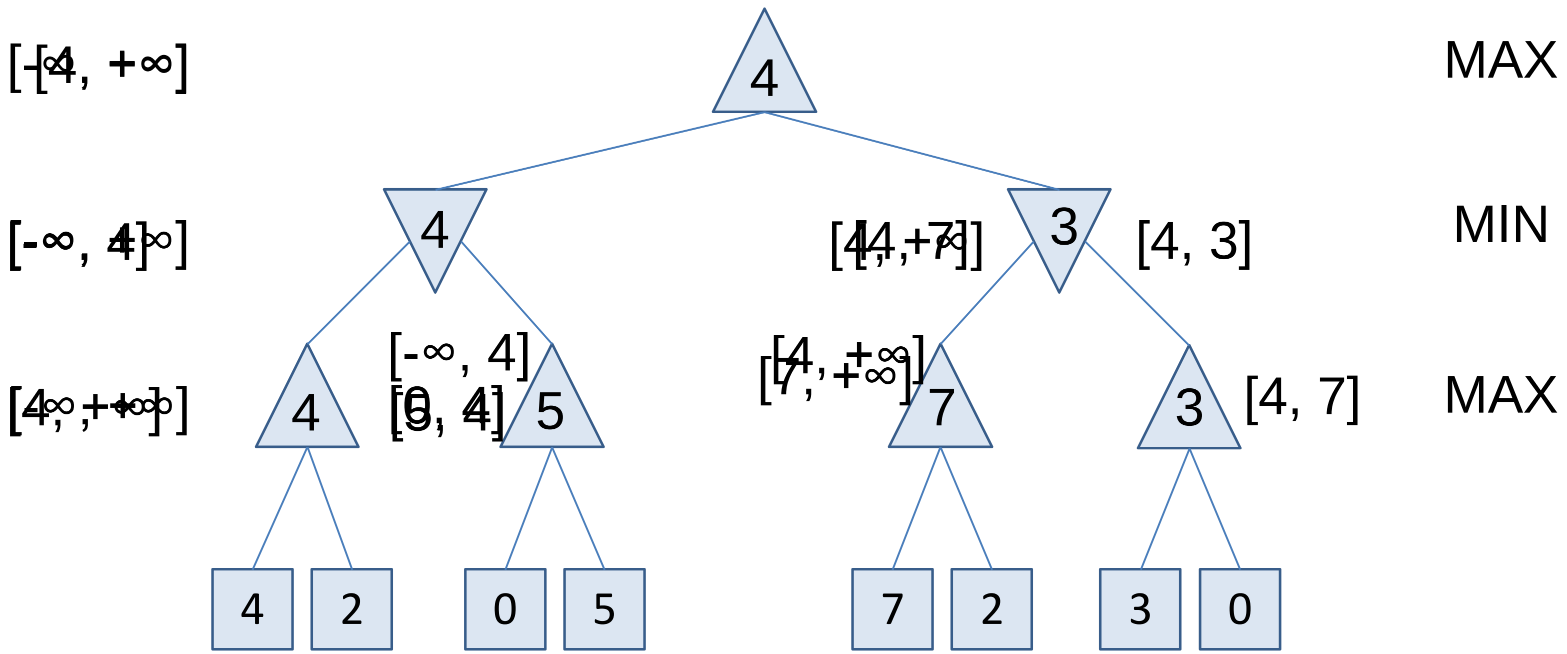
- what we really need at that moment is a bound (not the precise value)
- Remember Aspiration Search?
  - $x \leq \alpha_0$ - failing low (we know, that solution is $\leq x$)
  - $x \geq \beta_0$ - failing high (we know, that solution is $\geq x$)
- What if we use a null-window $[\alpha, \alpha+1]$ (or $[\alpha,\alpha]$)?
  - we obtain a bound …

NegaScout

function negascout(node, depth, α, β, Player)

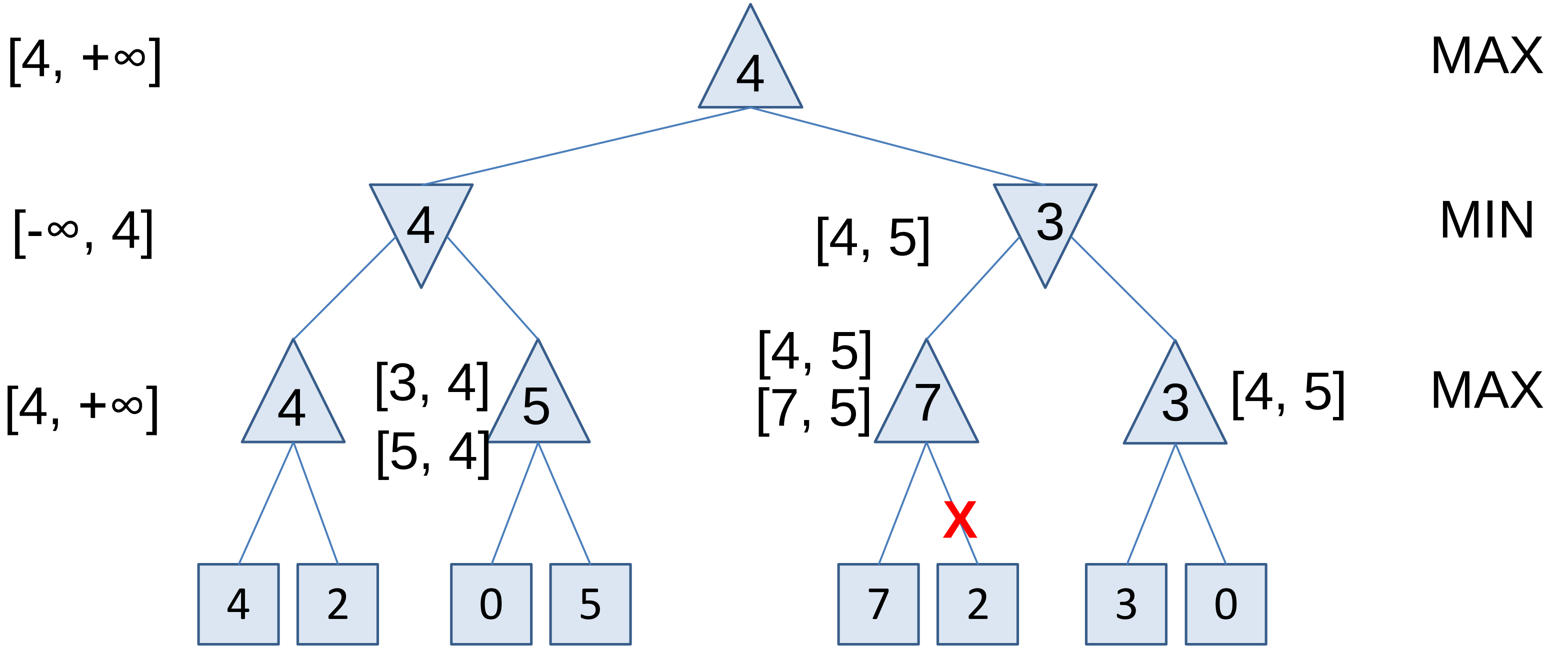
- **if** ((depth = 0) or (node is a terminal node)) **return** eval(node)
- b := β
- **for each** child of node
- v := -negascout(child, depth-1, -b, -α, switch(Player)))
- **if** ((α < v) and (child is not the first child))
- v := -negascout(child, depth-1, -β, -α, switch(Player)))
- α := max(α, v)
- **if** (β≤α) break
- b := α + 1
- **return** α

Alpha-Beta vs. Negascout



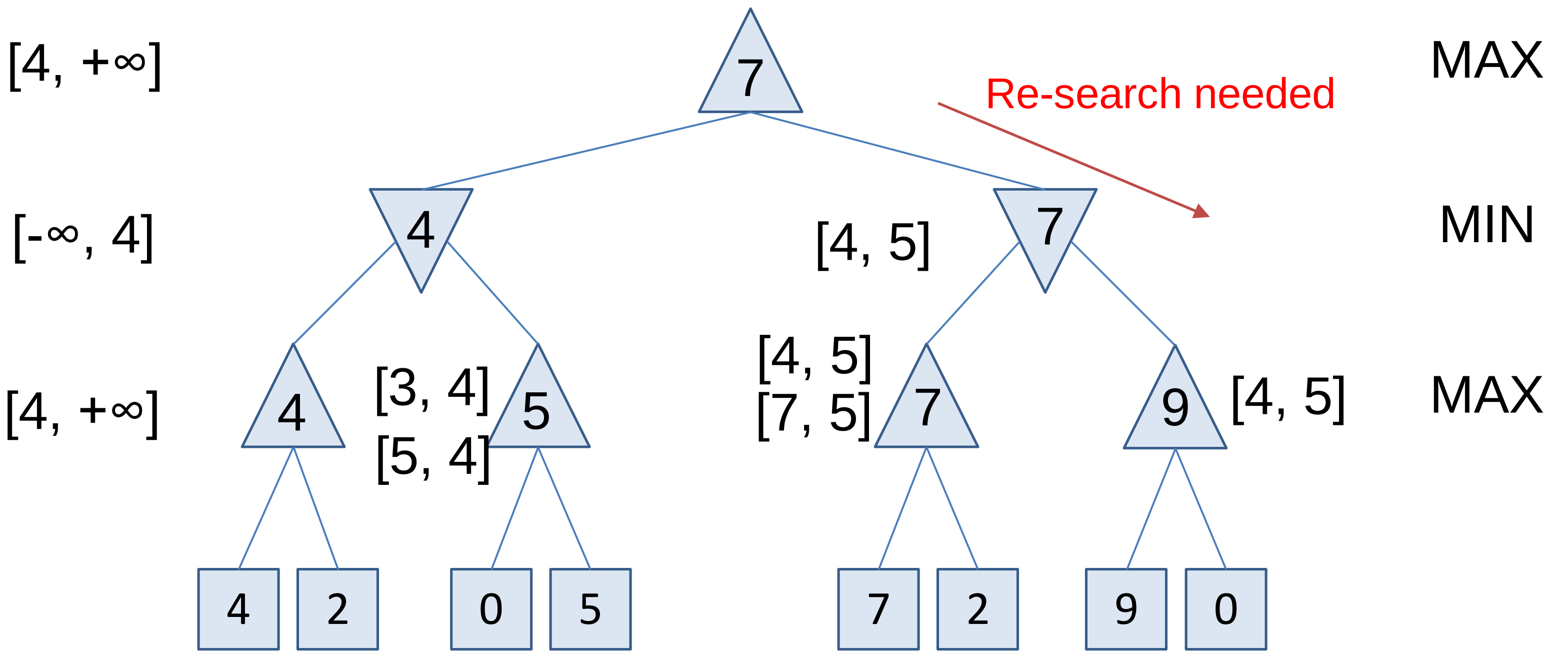
Scout idea and the change of intervals are demonstrated, the values are not multiplied by -1 in this example.

Alpha-Beta vs. Negascout



Scout idea and the change of intervals are demonstrated, the values are not multiplied by -1 in this example.

Alpha-Beta vs. Negascout



Scout idea and the change of intervals are demonstrated, the values are not multiplied by -1 in this example.

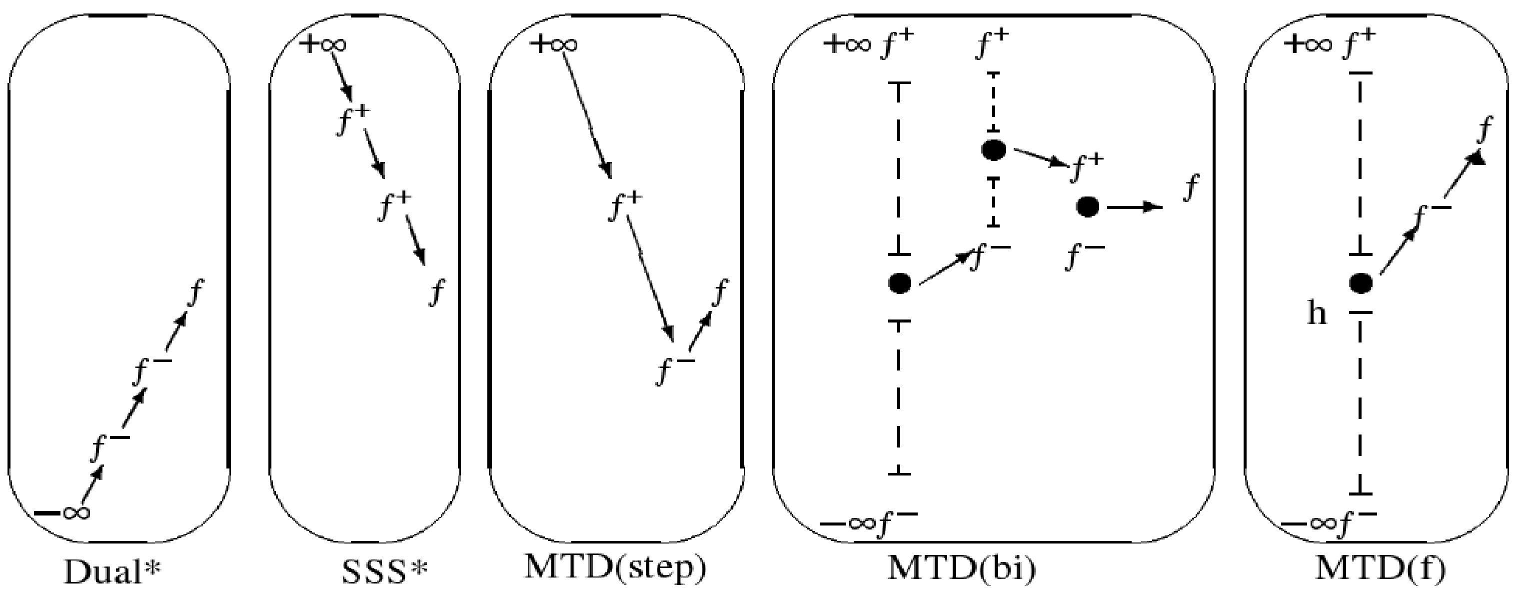
NegaScout

- also termed Principal Variation Search (PVS)
- dominates alpha-beta
  - never evaluates more **different nodes** than alpha-beta
  - can evaluate some nodes more than once

- depends on the move ordering
- can benefit from transposition tables (cache)
- generally 10-20% faster compared to alpha-beta

MTD

- Memory-enhanced Test Driver

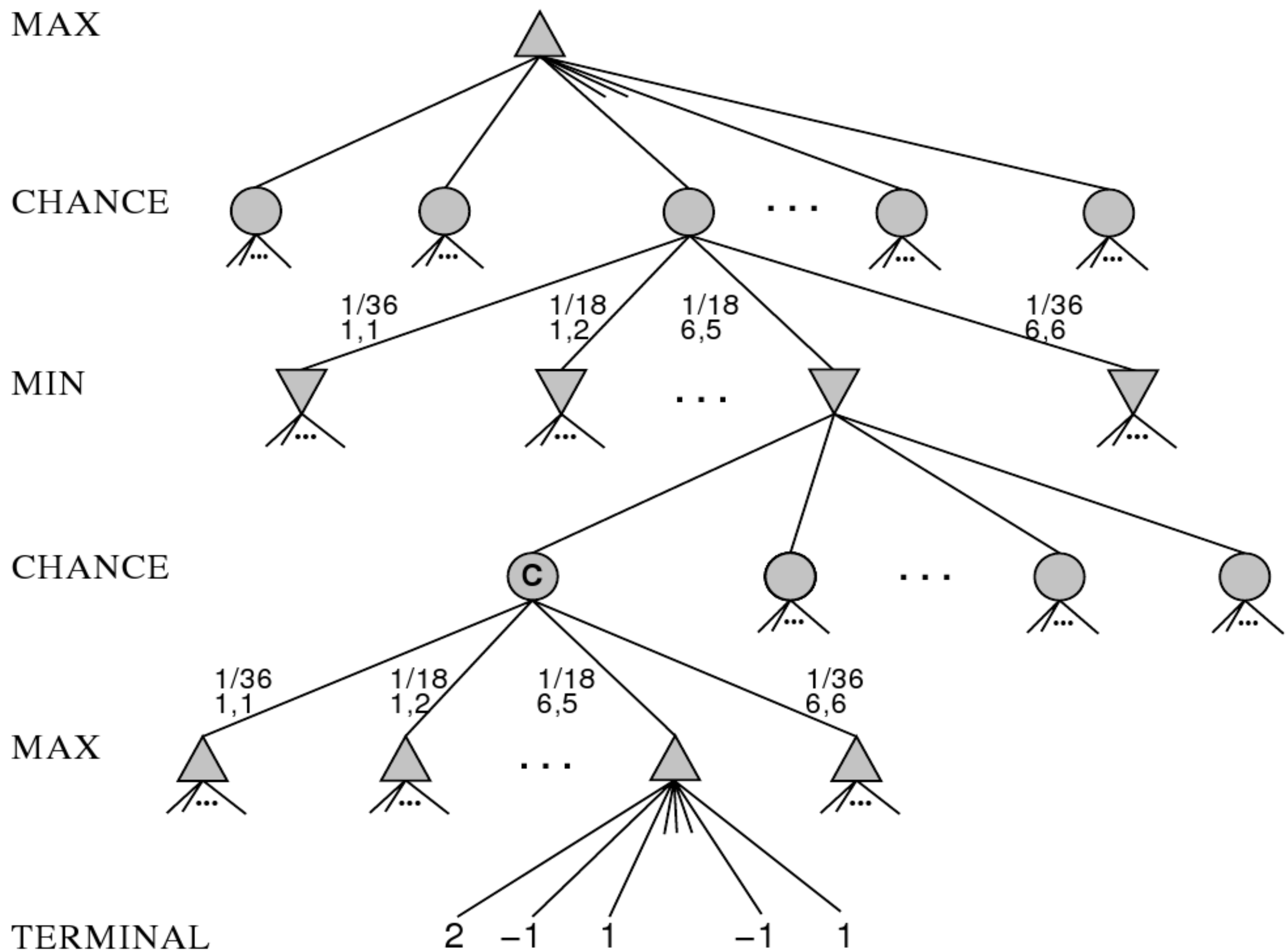


- Best-first fixed-depth minimax algorithms. Plaat et. al. , In *Artificial Intelligence,* Volume 87, Issues 1-2, November 1996, Pages 255-293

Further Topics

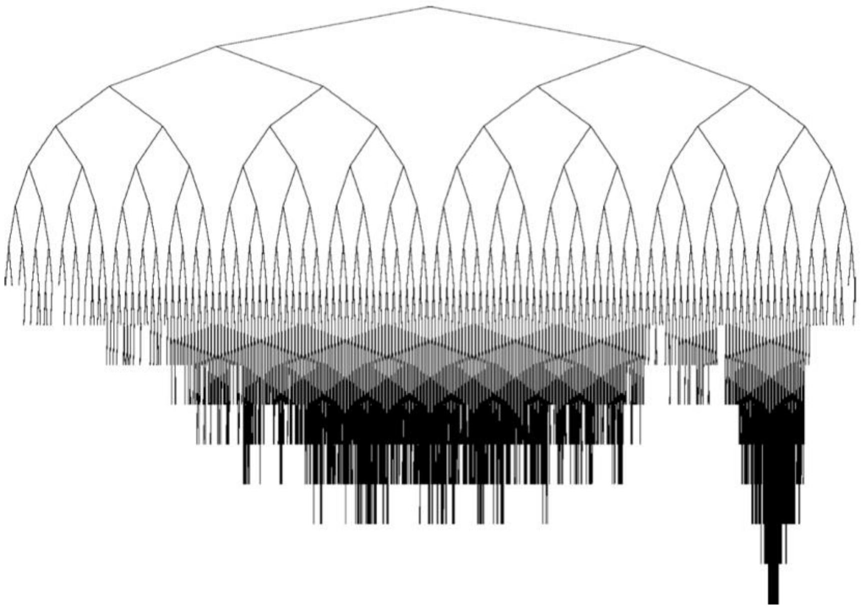
- more complex games
  - games with uncertainty
    - chance (Nature player), calculating expected utilities

Other Games - Chance nodes



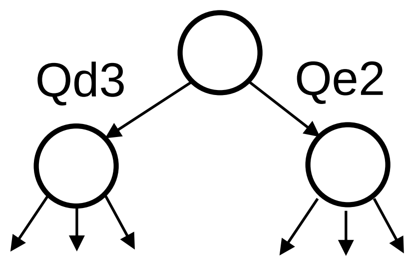
Towards more scalable algorithms …

- we do not want to evaluate all paths equally
- we want to search more deeply (thoroughly) more prospect variants
- we do not want to spend time with bad variants



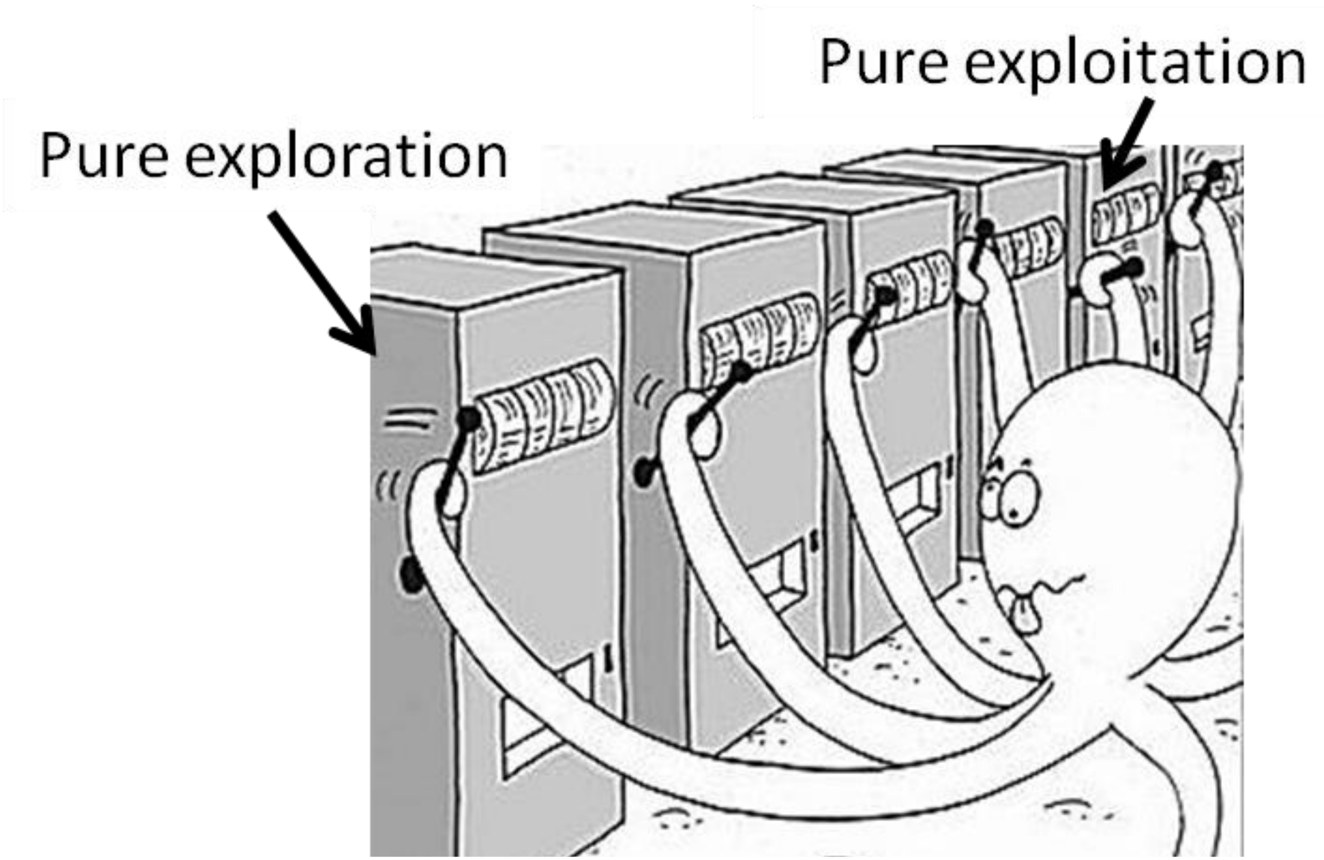
Towards more scalable algorithms …

- let's start from the beginning



- what if we estimate that Qd3 is (right now) a better move than Qe2
- there is a dilemma
  - either we want to get a better further plan (and thus also an estimate) of the better move (Qd3)
  - or we want to find a better continuation for the worse move (Qe2) – maybe there is one and we've just missed it before

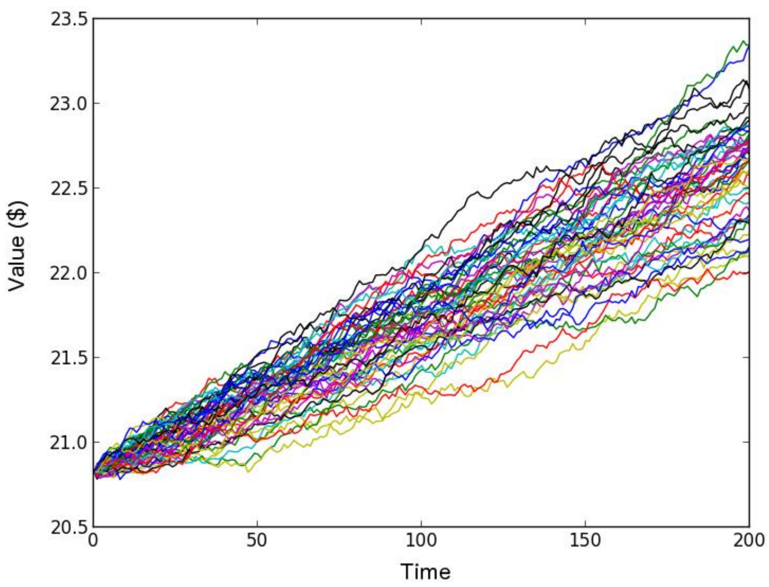
Exploration vs. Exploitation



Monte Carlo Methods

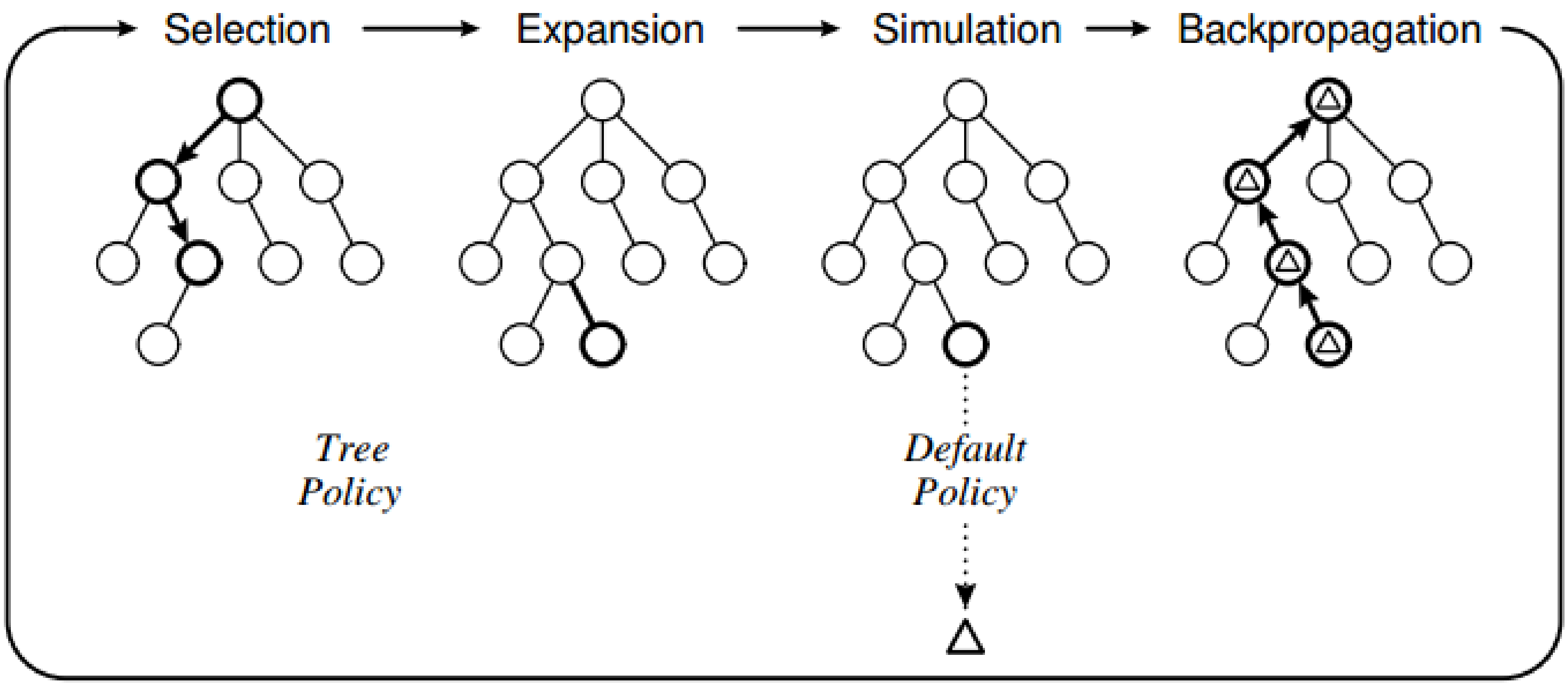
- what if we do not have evaluation function?
  - we can estimate the value of the position with a Monte Carlo method
  - from a given position we perform random samples until the terminal position of the game
  - the more samples we perform, the better estimate of the true value we get

Simulated paths of the value of an asset using Monte Carlo

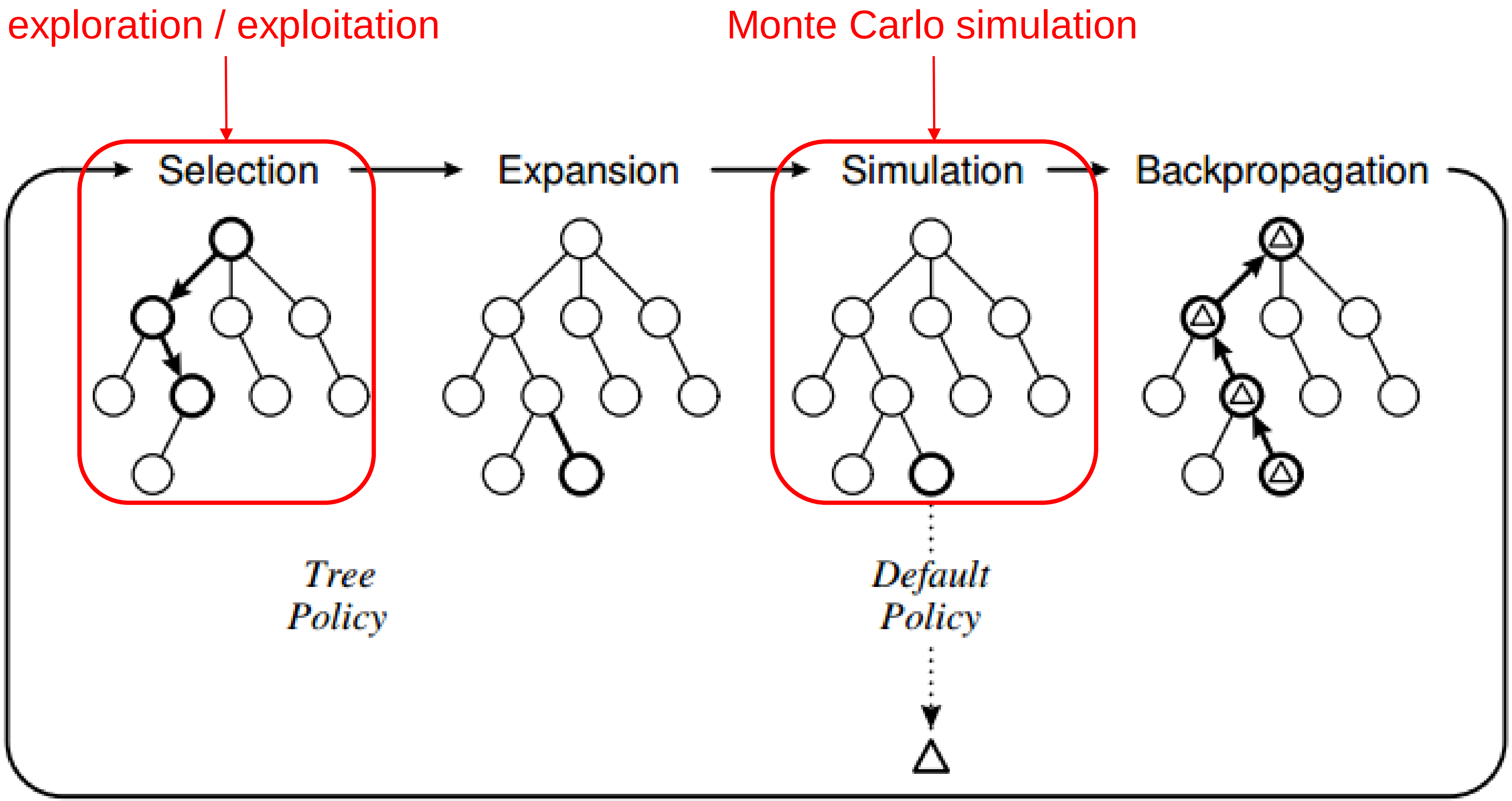


Monte Carlo Tree Search

putting it all together

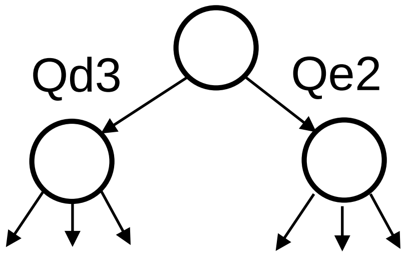


Monte Carlo Tree Search



Exploration vs. Exploitation

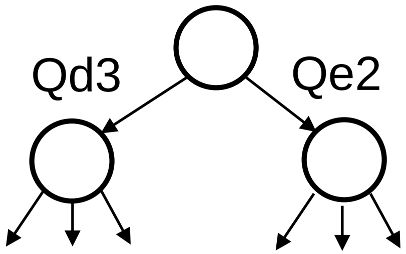
- bandit theory



- UCB – upper confidence bounds
  - $\arg\max_{v' \in children(v)} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

Exploration vs. Exploitation

- bandit theory



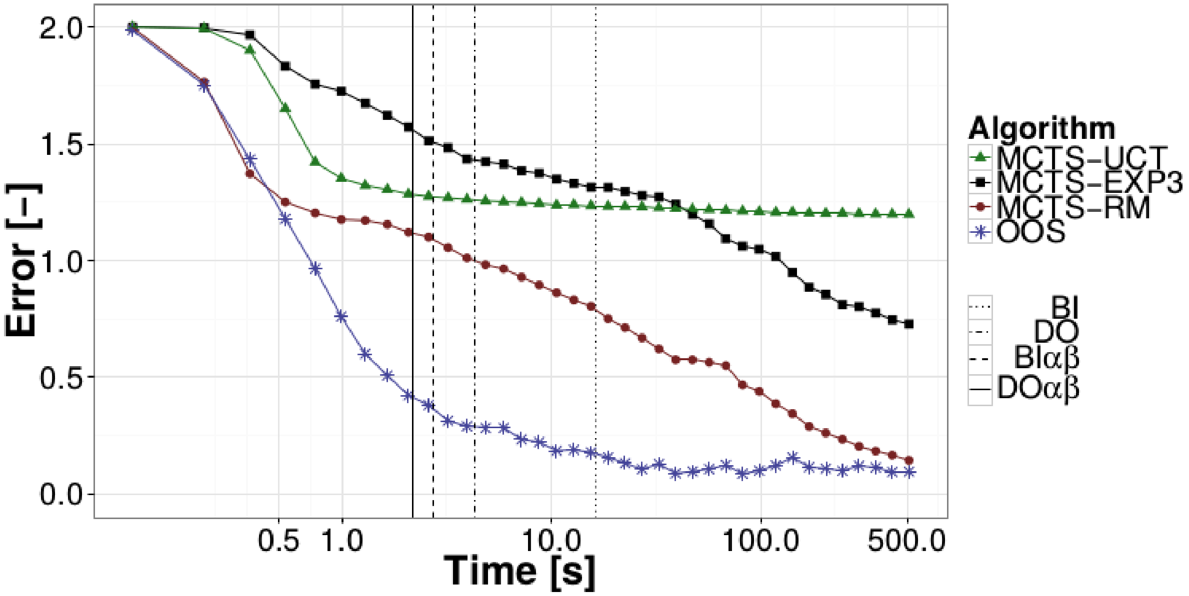
- UCB – upper confidence bounds
  - $\arg\max_{v' \in children(v)} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

average utility

exploration factor

Exploration vs. Exploitation

- many existing variants for the bandit problem
  - UCB1
  - EXP3
  - UCB-V
  - …
- can have a very different performance in practice



MCTS and Parameter Tuning

- Different bandit methods can have different parameters
- Practical performance depends on the correct choice
- The choice is domain dependent
- The choice is opponent dependent (!)

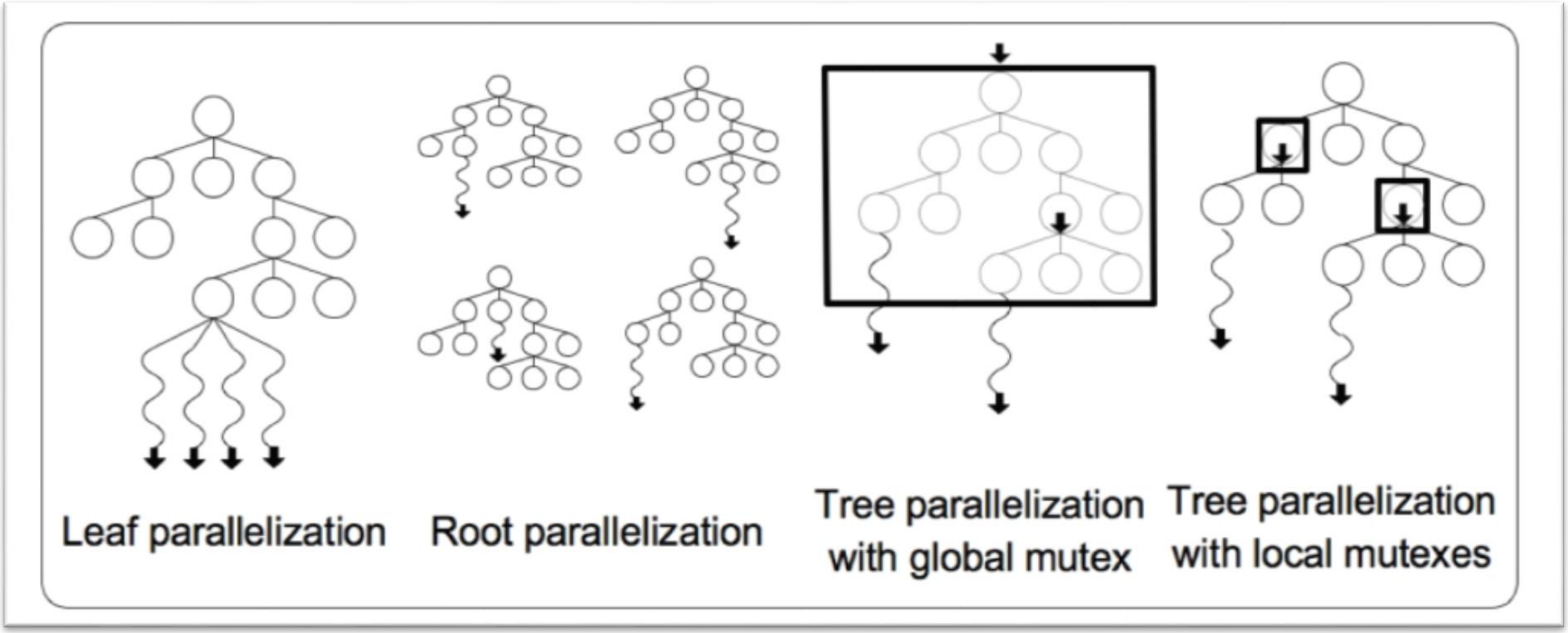
MCTS and Parameter Tuning

		DOαβ	OOS(0.6)	UCT(2)	EXP3(0.2)	RM(0.1)	Mean
OOS	0.5	35.3(2.9)	50.9(3.6)	28.5(3.3)	54.9(3.6)	43.7(3.5)	42.66
OOS	0.4	35.0(2.9)	56.0(3.6)	26.6(3.2)	56.1(3.6)	42.6(3.6)	43.26
OOS	**0.3**	36.5(3.0)	57.8(3.5)	27.7(3.2)	55.7(3.6)	44.8(3.6)	**44.5**
OOS	0.2	35.0(2.9)	53.1(3.6)	26.8(3.2)	54.1(3.6)	41.4(3.5)	42.08
OOS	0.1	34.6(2.9)	55.6(3.6)	24.1(3.1)	56.2(3.6)	43.0(3.6)	42.7
UCT	1.5	83.2(2.2)	74.0(3.8)	79.1(2.9)	87.4(2.9)	70.6(3.9)	78.86
UCT	1	83.8(2.1)	74.8(3.7)	81.4(2.7)	89.8(2.6)	68.8(4.0)	79.72
UCT	**0.8**	86.5(2.0)	77.9(3.6)	77.1(3.0)	89.2(2.7)	74.1(3.8)	**80.96**
UCT	0.6	89.4(1.8)	75.7(3.7)	54.9(3.9)	90.0(2.6)	74.1(3.7)	76.82
UCT	0.4	75.8(2.6)	75.0(3.7)	31.4(3.7)	89.8(2.6)	70.6(3.9)	68.52
EXP3	0.9	47.8(3.1)	68.2(2.8)	23.1(2.4)	67.2(2.8)	55.2(2.8)	52.3
EXP3	**0.8**	46.9(3.1)	68.4(3.6)	23.0(3.1)	74.2(3.4)	61.5(3.7)	**54.8**
EXP3	0.6	42.5(3.1)	67.6(3.7)	20.4(3.1)	65.4(3.7)	59.4(3.8)	51.06
EXP3	0.5	38.7(3.0)	60.9(3.8)	15.1(2.7)	64.7(3.7)	52.9(3.9)	46.46
EXP3	0.4	35.9(3.0)	57.5(3.9)	17.5(3.0)	64.1(3.8)	54.9(3.9)	45.98
RM	0.5	44.5(3.0)	41.1(3.5)	31.7(3.3)	49.4(3.6)	34.3(3.3)	40.2
RM	0.3	42.8(3.0)	52.1(3.5)	33.8(3.4)	61.2(3.5)	43.7(3.5)	46.72
RM	0.2	41.8(3.0)	55.7(3.6)	30.7(3.3)	59.2(3.5)	46.4(3.6)	46.76
RM	**0.1**	37.0(2.9)	58.1(3.5)	34.9(3.4)	57.6(3.6)	54.1(3.6)	**48.34**
RM	0.05	36.4(3.0)	59.6(3.5)	29.7(3.3)	59.3(3.5)	51.1(3.6)	47.22

Heuristics and MCTS

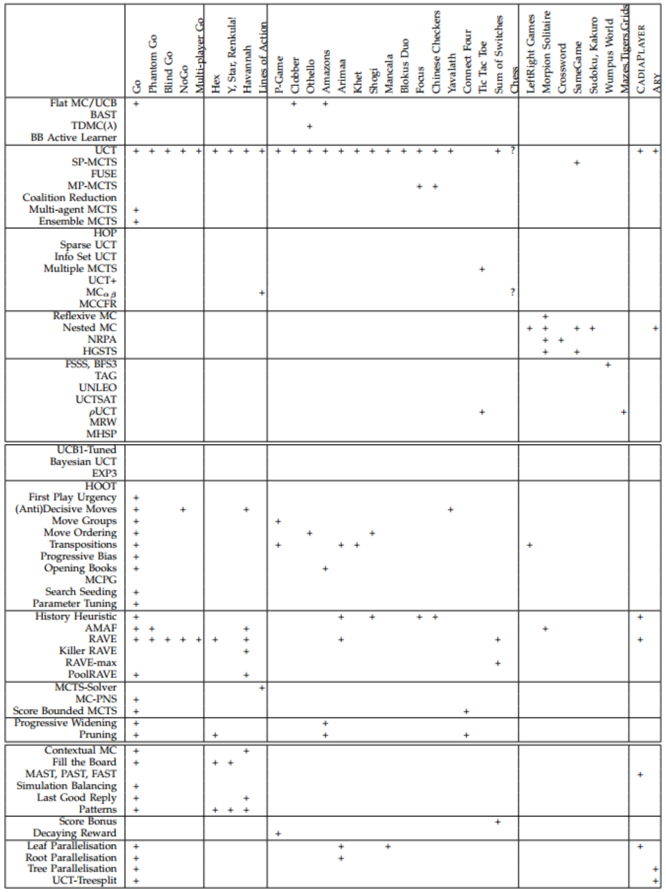
- there are several points where MCTS can benefit from domain-specific heuristic
  - progressive unpruning/widening
    - standard MCTS adds all children
  - heavy rollout simulations
    - simulations do not have to be completely random
    - tradeoff between bias and complexity vs. speed
  - using evaluation function instead of simulation
    - often combined with previous

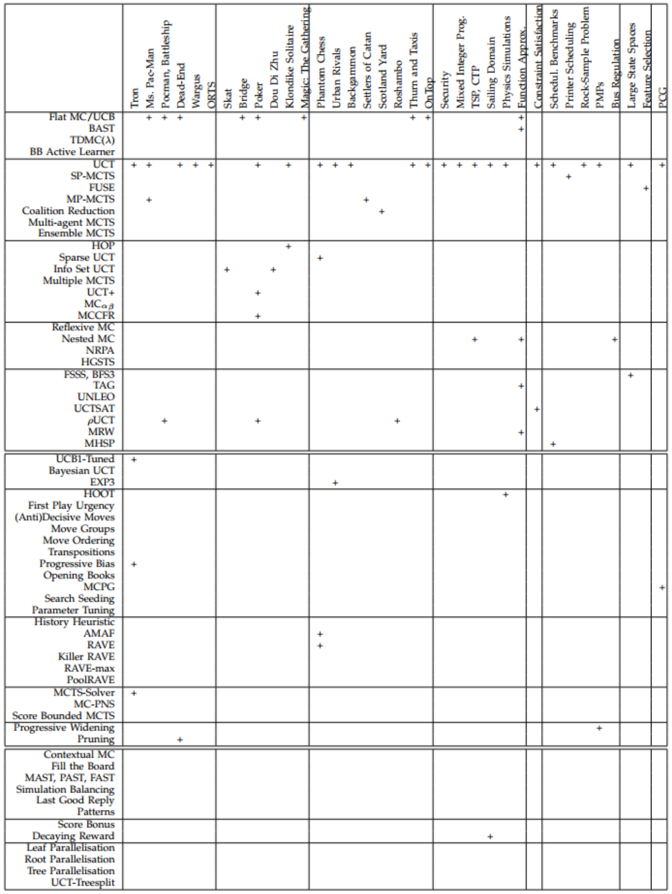
Parallelization of MCTS



Variants of MCTS

- there are **many** improvements and variants of MCTS
  - (see ``A Survey of Monte Carlo Tree Search Methods'' by Browne et al. 2012)





Games and Game Theory

- one shot simultaneous-move games
  - Rock-Paper-Scissors
- sequential games with simultaneous moves
  - Tron, many card games, …
  - alpha-beta algorithm can be generalized
- games with imperfect information

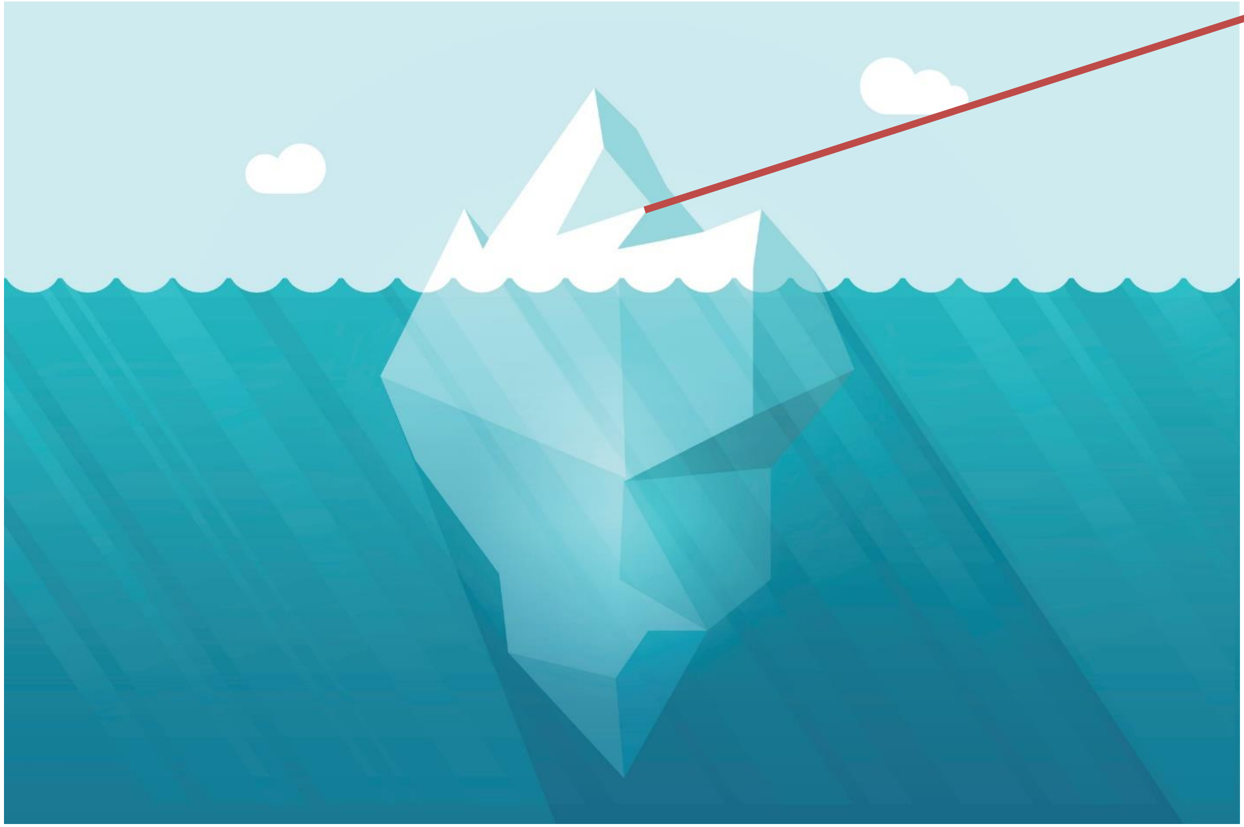
Games and Game Theory in AIC

attacks

protects

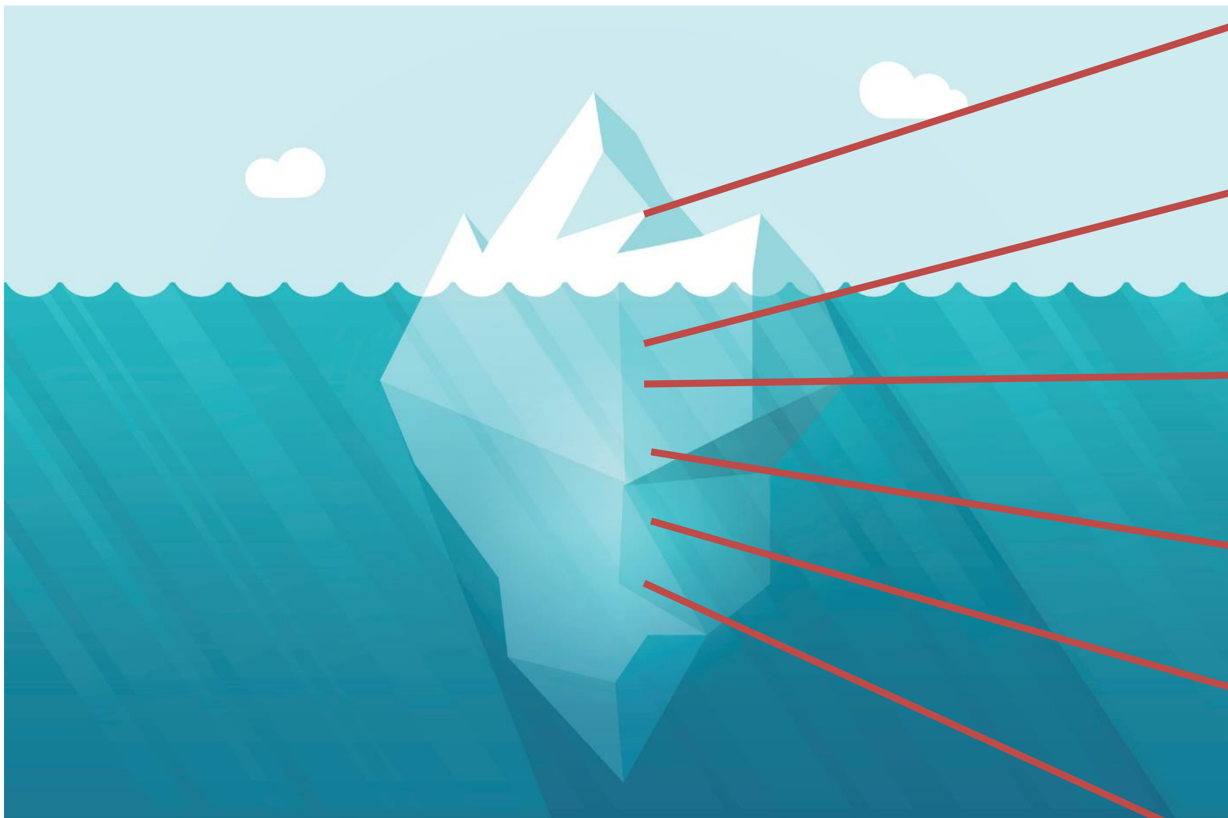
- Game theory has many possible applications
  - general algorithms for solving dynamic games with imperfect information
  - implementation of domain independent algorithms

Games and Game Theory in AIC



two-player games with perfect information

Games and Game Theory (in AIC)



two-player games with perfect information

two-player games with **imperfect information**

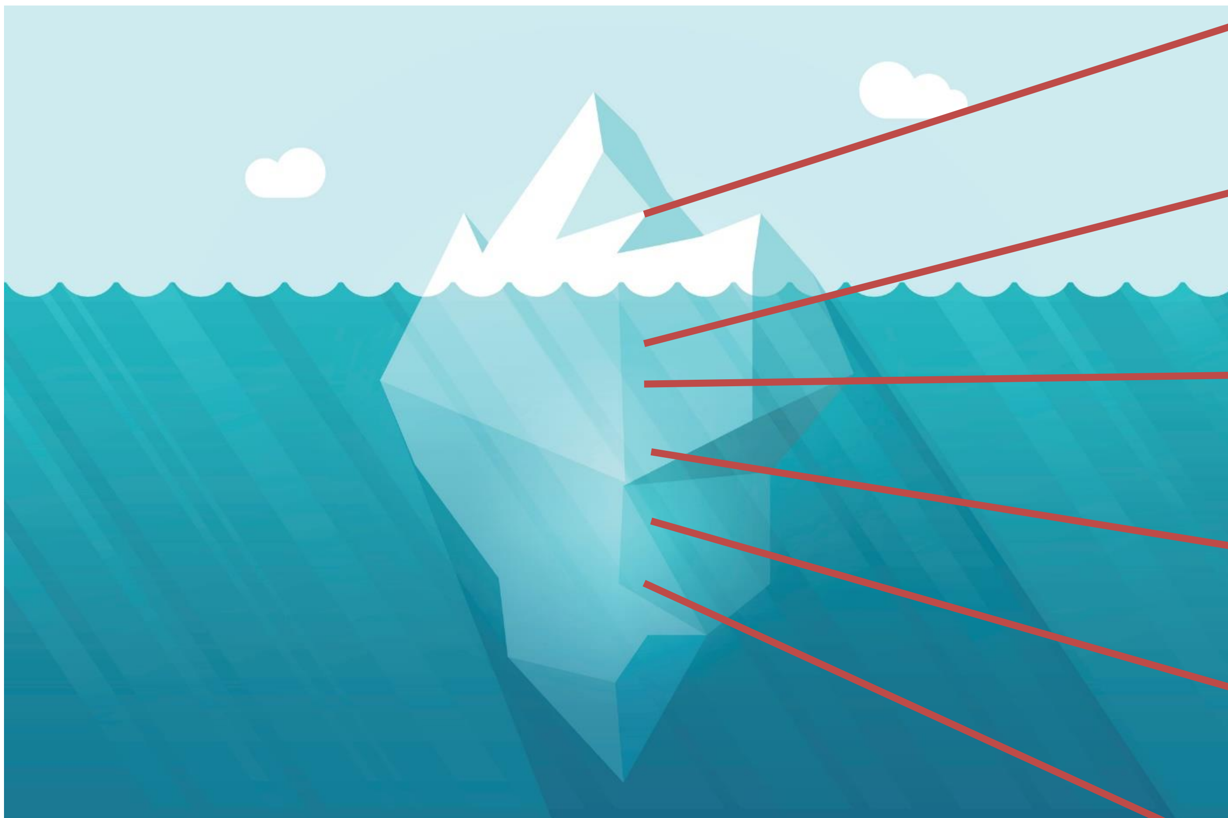
games with **infinite horizon** (stochastic games)

Game theory for **adversarial machine learning** (robust classifiers)

Multi-player games

Playing against human opponents **(bounded rationality)**

Games and Game Theory (in AIC)



two-player games with perfect information

two-player games with **imperfect information**

games with **infinite horizon** (stochastic games)

Game theory for **adversarial machine learning** (robust classifiers)

Multi-player games

Playing against human opponents **(bounded rationality)**

B4M36MAS – Multiagent Systems

XEP36AGT – Algorithmic Game Theory

Bachelor's project, Diploma thesis, …

http://aic.fel.cvut.cz/gametheory